Learning Regular Languages over Large Alphabets

Oded Maler and Irini Eleftheria Mens

CNRS-VERIMAG
University of Grenoble
France

Abstract. This work is concerned with regular languages defined over large alphabets, either infinite or just too large to be expressed enumeratively. We define a generic model where transitions are labeled by elements of a finite partition of the alphabet. We then extend Angluin's L^* algorithm for learning regular languages from examples for such automata. We have implemented this algorithm and we demonstrate its behavior where the alphabet is the set of natural numbers.

1 Introduction

The main contribution of this paper is a generic algorithm for learning regular languages defined over a large alphabet Σ. Such an alphabet can be infinite, like $\mathbb{N}$ or $\mathbb{R}$ or just so large, like $\mathbb{B}^n$ for very large n, that it is impossible or impractical to treat it in an enumerative way, that is, to write down $\delta(q,a)$ for every $a \in \Sigma$. The obvious solution is to use a *symbolic* representation where transitions are labeled by predicates which are applicable to the alphabet in question. Learning algorithms infer an automaton from a finite set of words (the *sample*) for which membership is known. Over small alphabets, the sample should include the set S all the shortest words that lead to each state and, in addition, the set $S \cdot \Sigma$ of all their Σ-continuations. Over large alphabets this is not a practical option and as an alternative we develop a symbolic learning algorithm over *symbolic words* which are only partially backed up by the sample. In a sense, our algorithm is a combination of automaton learning and learning of non-temporal functions. Before getting technical, let us discuss briefly some motivation.

Finite automata are among the corner stones of Computer Science. From a practical point of view they are used daily in various domains ranging from syntactic analysis, design of user interfaces or administrative procedures to implementation of digital hardware and verification of software and hardware protocols. Regular languages admit a very nice, clean and comprehensive theory where different formalisms such as automata, logic, regular expressions, semigroups and grammars are shown to be equivalent. As for learning from examples, a problem introduced by Moore [Moo56], the Nerode right-congruence relation [Ner58] which declares two *input histories* as equivalent if they lead to the same *future continuations*, provides a crisp characterization of what a *state* in a dynamical system is in terms of observable input-output behavior. All algorithms for learning automata from examples, starting with the seminal work of Gold [Gol72] and culminating in the well-known L^* algorithm of Angluin [Ang87] are based on this concept [DlH10].

One weakness, however, of the classical theory of regular languages is that it is rather "thin" and "flat". In other words, the alphabet is often considered as a small set devoid of any additional structure. On such alphabets, classical automata are good for expressing and exploring the temporal (sequential, monoidal) dimension embodied by the concatenation operations, but less good in expressing "horizontal" relationships. To make this statement more concrete, consider the verification of a system consisting of n automata running in parallel, making independent as well as synchronized transitions. To express the set of joint behaviors of this product of automata as a formal language, classical theory will force you to use the exponential alphabet of global states and indeed, a large part of verification is concerned with fighting this explosion using constructs such as BDDs and other logical forms that exploit the sparse interaction among components. This is done, however, without a real interaction with classical formal language theory (one exception is the theory of *traces* [DR95] which attempts to treat this issue but in a very restricted context).[1]

These and other considerations led us to use *symbolic automata* as a generic framework for recognizing languages over large alphabets where transitions outgoing from a state are labeled, semantically speaking, by *subsets* of the alphabet. These subsets are expressed syntactically according to the specific alphabet used: Boolean formulae when $\Sigma = \mathbb{B}^n$ or by some classes of inequalities when $\Sigma = \mathbb{N}$. Determinism and completeness of the transition relation, which are crucial for learning and minimization, can be enforced by requiring that the subsets of Σ that label the transitions outgoing from a given state form a *partition* of the alphabet.

Readers working on program verification or hybrid automata are, of course, aware of automata with symbolic transition guards but it should be noted that in our model *no auxiliary variables* are added to the automaton. Let us stress this point by looking at a popular extension of automata to infinite alphabets, initiated by Kaminski and Francez [KF94] using *register automata* to accept *data languages* (see [BLP10] for theoretical properties and [HSJC12] for learning algorithms). In that framework, the automaton is augmented with additional registers that can store some input letters. The registers can then be compared with newly-read letters and influence transitions. With register automata one can express, for example, the requirement that your password at login is the same as the password at sign-up. This very restricted use of memory makes register automata much simpler than more notorious automata with variables whose emptiness problem is typically undecidable. The downside is that beyond *equality* they do not really exploit the potential richness of the alphabets/theories.

Our approach is different: we do allow the *values* of the input symbols to influence transitions via predicates, possibly of a restricted complexity. These predicates involve domain *constants* and they partition the alphabet into finitely many classes. For example, over the integers a state may have transitions labeled by conditions of the form $c_1 \leq x \leq c_2$ which give real (but of limited resolution) access to the input domain. On the other hand, we insist on a finite (and small) memory so that the exact value of x *cannot* be registered and has no future influence beyond the transition it has triggered. The *symbolic transducers*, recently introduced by [VHL$^+$12], are based on the same

[1] This might also be the reason that Temporal Logic is more popular in verification than regular expressions because the nature of *until* is less global and less synchronous than concatenation.

principle. Many control systems, artificial (sequential machines working on quantized numerical inputs) as well as natural (central nervous system, the cell), are believed to operate in this manner.

We then develop a symbolic version of Angluin's L^* algorithm for learning regular sets from queries and counter-examples whose output is a symbolic automaton. The main difference relative to the concrete algorithm is that in the latter, every transition $\delta(q, a)$ in a conjectured automaton has at least one word in the sample that exercises it. In the symbolic case, a transition $\delta(q, \boldsymbol{a})$ where $\boldsymbol{a}$ is a *set* of concrete symbols, will be backed up in the sample only by a *subset* of $\boldsymbol{a}$. Thus, unlike concrete algorithms where a counter-example always leads to a discovery of one or more new states, in our algorithm it may sometimes only modify the boundaries between partition blocks without creating new states.

The rest of the paper is organized as follows. In Section 2 we provide a quick summary of learning algorithms over small alphabets. In Section 3 we define symbolic automata and then extend the structure which underlies all automaton learning algorithms, namely the *observation table*, to be symbolic, where symbolic letters represent sets, and where entries in the table are supported only by partial evidence. In Section 4 we write down a symbolic learning algorithm and illustrate the behavior of a prototype implementation on learning subsets of $\mathbb{N}^*$. We conclude by a discussion of past and future work.

2 Learning Concrete Automata

We briefly survey Angluin's L^* algorithm [Ang87] for learning regular sets from membership queries and counter-examples, with slightly modified definitions to accommodate for its symbolic extension. Let Σ be a finite alphabet and let Σ^* be the set of sequences (words) over Σ. Any order relation $<$ over Σ can be naturally lifted to a lexicographic order over Σ^*. With a language $L \subseteq \Sigma^*$ we associate a *characteristic function* $f : \Sigma^* \to \{0, 1\}$.

A *deterministic finite automaton* over Σ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, where Q is a non-empty finite set of *states*, $q_0 \in Q$ is the *initial* state, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $F \subseteq Q$ is the set of *final* or *accepting* states. The transition function δ can be extended to $\delta : Q \times \Sigma^* \to Q$, where $\delta(q, \epsilon) = q$ and $\delta(q, u \cdot a) = \delta(\delta(q, u), a)$ for $q \in Q$, $a \in \Sigma$ and $u \in \Sigma^*$. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if $\delta(q_0, w) \in F$, otherwise w is *rejected*. The language recognized by $\mathcal{A}$ is the set of all accepted words and is denoted by $L(\mathcal{A})$.

Learning algorithms, represented by the *learner*, are designed to infer an unknown regular language L (the *target language*). The learner aims to construct a finite automaton that recognizes the target language by gathering information from the *teacher*. The *teacher* knows the target language and can provide information about it. It can answer two types of queries: *membership queries*, i.e., whether a word belongs to the target language, and *equivalence queries*, i.e., whether a conjectured automaton suggested by the learner is the right one. If this automaton fails to accept L the teacher responds to the equivalence query by a *counter-example*, a word misclassified by the conjectured automaton.

In the L^* algorithm, the learner starts by asking membership queries. All information provided is suitably gathered in a table structure, the *observation table*. Then, when the information is sufficient, the learner constructs a *hypothesis automaton* and poses an equivalence query to the teacher. If the answer is positive then the algorithm terminates and returns the conjectured automaton. Otherwise the learner accommodates the information provided by the counter-example into the table, asks additional membership queries until it can suggest a new hypothesis and so on, until termination.

A prefix-closed set $S \uplus R \subset \Sigma^*$ is a *balanced Σ-tree* if $\forall a \in \Sigma$: 1) For every $s \in S$ $s \cdot a \in S \cup R$, and 2) For every $r \in R$, $r \cdot a \notin S \cup R$. Elements of R are called *boundary elements* or *leaves*.

Definition 1 (Observation Table). *An* observation table *is a tuple $T = (\Sigma, S, R, E, f)$ such that Σ is an alphabet, $S \cup R$ is a finite balanced Σ-tree, E is a subset of Σ^* and $f : (S \cup R) \cdot E \to \{0, 1\}$ is the classification function, a restriction of the characteristic function of the target language L.*

The set $(S \cup R) \cdot E$ is the *sample* associated with the table, that is, the set of words whose membership is known. The elements of S admit a tree structure isomorphic to a *spanning tree* of the transition graph rooted in the initial state. Each $s \in S$ corresponds to a state q of the automaton for which s is an *access sequence*, one of the shortest words that lead from the initial state to q. The elements of R should tell us about the back- and cross-edges in the automaton and the elements of E are "experiments" that should be sufficient to distinguish between states. This works by associating with every $s \in S \cup R$ a specialized classification function $f_s : E \to \{0, 1\}$, defined as $f_s(e) = f(s \cdot e)$, which characterizes the row of the observation table labeled by s. To build an automaton from a table it should satisfy certain conditions.

Definition 2 (Closed, Reduced and Consistent Tables). *An observation table T is:*

- *Closed if for every $r \in R$, there exists an $s \in S$, such that $f_r = f_s$;*
- *Reduced if for every $s, s' \in S$ $f_s \neq f_{s'}$;*
- *Consistent if for every $s, s' \in S$, $f_s = f_{s'}$ implies $f_{s \cdot a} = f_{s' \cdot a}, \forall a \in \Sigma$.*

Note that a reduced table is trivially consistent and that for a closed and reduced table we can define a function $g : R \to S$ mapping every $r \in R$ to the unique $s \in S$ such that $f_s = f_r$. From such an observation table $T = (\Sigma, S, R, E, f)$ one can construct an automaton $\mathcal{A}_T = (\Sigma, Q, q_0, \delta, F)$ where $Q = S$, $q_0 = \epsilon$, $F = \{s \in S : f_s(\epsilon) = 1\}$ and

$$\delta(s, a) = \begin{cases} s \cdot a & \text{when } s \cdot a \in S \\ g(s \cdot a) & \text{when } s \cdot a \in R \end{cases}$$

The learner attempts to keep the table closed at all times. The table is not closed when there is some $r \in R$ such that f_r is different from f_s for all $s \in S$. To close the table, the learner moves r from R to S and adds the Σ-successors of r to R. The extended table is then filled up by asking membership queries until it becomes closed.

Variants of the L^* algorithm differ in the way they treat counter-examples, as described in more detail in [BR04]. The original algorithm [Ang87] adds all the *prefixes* of the counter-example to S and thus possibly creating inconsistency that should be fixed.

The version proposed in [MP95] for learning ω-regular languages adds all the *suffixes* of the counter-example to E. The advantage of this approach is that the table always remains consistent and reduced with S corresponding exactly to the set of states. A disadvantage is the possible introduction of redundant columns that do not contribute to further discrimination between states. The symbolic algorithm that we develop in this paper is based on an intermediate variant, referred to in [BR04] as the *reduced observation algorithm*, where some prefixes of the counter-example are added to S and some suffixes are added to E.

Example: We illustrate the behavior of the L^* algorithm while learning $L = a\Sigma^*$ over $\Sigma = \{a, b\}$. We use $+w$ to indicate a counter-example $w \in L$ rejected by the conjectured automaton, and $-w$ for the opposite case. Initially, the observation table is $T_0 = (\Sigma, S, R, E, f)$ with $S = E = \{\epsilon\}$ and $R = \Sigma$ and we ask membership queries for all words in $(S \cup R) \cdot E = \{\epsilon, a, b\}$ to obtain table T_0, shown in Fig. 1. The table is not closed so we move a to S, add its continuations, aa and ab to R and ask membership queries to obtain the closed table T_1, from which the hypothesis automaton $\mathcal{A}_1$ of Fig. 2 is derived. In response to the equivalence query for $\mathcal{A}_1$, a counter-example $-ba$ is presented, its prefixes b and ba are added to S and their successors are added to R, resulting in table T_2 of Fig. 1. This table is not consistent: two elements ϵ and b in S are equivalent but their a-successors a and ba are not. Adding a to E and asking membership queries yields a consistent table T_3 whose automaton $\mathcal{A}_3$ is the minimal automaton recognizing L. ⌟

T_0

	ϵ
ϵ	0
a	1
b	0

T_1

	ϵ
ϵ	0
a	1
b	0
aa	1
ab	1

T_2

	ϵ
ϵ	0
a	1
b	0
ba	0
aa	1
ab	1
bb	0
baa	0
bab	0

T_3

	ϵ	a
ϵ	0	1
a	1	1
b	0	0
ba	0	0
aa	1	1
ab	1	1
bb	0	0
baa	0	0
bab	0	0

Fig. 1. Observation tables of L^* while learning $a \cdot \Sigma^*$.

3 Symbolic Automata

Symbolic automata are automata over large alphabets where from each state there is a small number of outgoing transitions labelled by subsets of Σ that form a *partition* of the alphabet. Let Σ be a large and possibly infinite alphabet, that we call the *concrete* alphabet. Let ψ be a *total surjective* function from Σ to a finite (symbolic) alphabet $\boldsymbol{\Sigma}$. For each *symbolic letter* $\boldsymbol{a} \in \boldsymbol{\Sigma}$ we assign a Σ-semantics $[\boldsymbol{a}]_\psi = \{a \in \Sigma : \psi(a) = \boldsymbol{a}\}$.

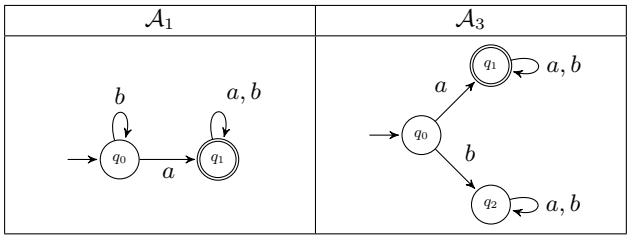

Fig. 2. Hypothesis automata for $a \cdot \Sigma^*$

Since ψ is total and surjective, the set $\{[\boldsymbol{a}]_\psi : \boldsymbol{a} \in \boldsymbol{\Sigma}\}$ forms a *partition* of Σ. We will often omit ψ from the notation and use $[\boldsymbol{a}]$ where ψ, which is always present, is clear from the context. The Σ-semantics can be extended to symbolic words of the form $\boldsymbol{w} = \boldsymbol{a}_1 \cdot \boldsymbol{a}_2 \cdots \boldsymbol{a}_k \in \boldsymbol{\Sigma}^*$ as the concatenation of the concrete one-letter languages associated with the respective symbolic letters or, recursively speaking, $[\boldsymbol{\epsilon}] = \{\epsilon\}$ and $[\boldsymbol{w} \cdot \boldsymbol{a}] = [\boldsymbol{w}] \cdot [\boldsymbol{a}]$ for $\boldsymbol{w} \in \boldsymbol{\Sigma}^*$, $\boldsymbol{a} \in \boldsymbol{\Sigma}$.

Definition 3 (Symbolic Automaton). *A* deterministic symbolic automaton *is a tuple* $\boldsymbol{\mathcal{A}} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$*, where*

- Σ *is the input alphabet,*
- $\boldsymbol{\Sigma}$ *is a finite alphabet, decomposable into* $\boldsymbol{\Sigma} = \biguplus_{q \in Q} \boldsymbol{\Sigma}_q$*,*
- $\psi = \{\psi_q : q \in Q\}$ *is a family of total surjective functions* $\psi_q : \Sigma \to \boldsymbol{\Sigma}_q$*,*
- Q *is a finite set of states,*
- $\boldsymbol{\delta} : Q \times \boldsymbol{\Sigma} \to Q$ *is a partial transition function decomposable into a family of total functions* $\boldsymbol{\delta}_q : \{q\} \times \boldsymbol{\Sigma}_q \to Q$*,*
- q_0 *is the initial state and* F *is the set of accepting states.*

Automaton $\boldsymbol{\mathcal{A}}$ can be viewed as representing a concrete deterministic automaton $\mathcal{A}$ whose transition function is defined as $\delta(q, a) = \boldsymbol{\delta}(q, \psi_q(a))$ and its accepted concrete language is $L(\boldsymbol{\mathcal{A}}) = L(\mathcal{A})$.

Remark: The association of a *symbolic language* with a symbolic automaton is more subtle because we allow different partitions of Σ and hence different input alphabets at different states, rendering the transition function *partial* with respect to $\boldsymbol{\Sigma}$. When in a state q and reading a symbol $\boldsymbol{a} \notin \boldsymbol{\Sigma}_q$, the transition to be taken is well defined only when $[\boldsymbol{a}] \subseteq [\boldsymbol{a'}]$ for some $\boldsymbol{a'} \in \boldsymbol{\Sigma}_q$. The model can, nevertheless, be made deterministic and complete over a refinement of the symbolic alphabet. Let

$$\boldsymbol{\Sigma'} = \prod_{q \in Q} \boldsymbol{\Sigma}_q, \text{ with the } \Sigma\text{-semantics } [(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)] = [\boldsymbol{a_1}] \cap \ldots \cap [\boldsymbol{a}_n]$$

and let $\tilde{\boldsymbol{\Sigma}} = \{\boldsymbol{b} \in \boldsymbol{\Sigma'} : [\boldsymbol{b}] \neq \emptyset\}$. We can then define an ordinary automaton $\widetilde{\boldsymbol{\mathcal{A}}} = (\tilde{\boldsymbol{\Sigma}}, Q, \tilde{\delta}, q_0, F)$ where, by construction, for every $\boldsymbol{b} \in \tilde{\boldsymbol{\Sigma}}$ and every $q \in Q$, there is $\boldsymbol{a} \in \boldsymbol{\Sigma}_q$ such that $[\boldsymbol{b}] \subseteq [\boldsymbol{a}]$ and hence one can define the transition function as $\tilde{\boldsymbol{\delta}}(q, \boldsymbol{b}) = \boldsymbol{\delta}(q, \boldsymbol{a})$. This model is more comfortable for language-theoretic studies but we stick in this paper to Definition 3 as it is more efficient. A similar choice has been made in [IHS13]. ⌟

Proposition 1 (Closure under Boolean Operations). *Languages accepted by deterministic symbolic automata are closed under Boolean operations.*

Proof. Closure under negation is immediate by complementing the set of accepting states. For intersection we adapt the standard product construction as follows. Let L_1, L_2 be languages recognized by the symbolic automata $\boldsymbol{\mathcal{A}}_1 = (\Sigma, \boldsymbol{\Sigma}_1, \psi_1, Q_1, \boldsymbol{\delta}_1, q_{01}, F_1)$ and $\boldsymbol{\mathcal{A}}_2 = (\Sigma, \boldsymbol{\Sigma}_2, \psi_2, Q_2, \boldsymbol{\delta}_2, q_{02}, F_2)$, respectively. Let $\boldsymbol{\mathcal{A}} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$, where

- $Q = Q_1 \times Q_2$, $q_0 = (q_{01}, q_{02})$, $F = F_1 \times F_2$
- For every $(q_1, q_2) \in Q$
  - $\boldsymbol{\Sigma}_{(q_1,q_2)} = \{(\boldsymbol{a}_1, \boldsymbol{a}_2) \in \boldsymbol{\Sigma}_1 \times \boldsymbol{\Sigma}_2 \mid [\boldsymbol{a}_1] \cap [\boldsymbol{a}_2] \neq \emptyset\}$
  - $\psi_{(q_1,q_2)}(a) = (\psi_{1,q_1}(a), \psi_{2,q_2}(a))\ \forall a \in \Sigma$
  - $\boldsymbol{\delta}((q_1, q_2), (\boldsymbol{a}_1, \boldsymbol{a}_2)) = (\boldsymbol{\delta}_1(q_1, \boldsymbol{a}_1), \boldsymbol{\delta}_2(q_2, \boldsymbol{a}_2))\ \forall (\boldsymbol{a}_1, \boldsymbol{a}_2) \in \boldsymbol{\Sigma}_{(q_1,q_2)}$

It is sufficient to observe that the corresponding implied concrete automata $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}$ satisfy $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ and the standard proof that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ follows. ∎

Equivalence queries are implemented by constructing a product automaton which accepts the symmetric difference between the target language and the conjectured one, finding shortest paths to accepting states and selecting a lexicographically minimal word.

Definition 4 (Balanced Symbolic Σ-Tree). *A balanced symbolic Σ-tree is a tuple $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi)$ where*

- *$\boldsymbol{S} \uplus \boldsymbol{R}$ is a prefix-closed subset of $\boldsymbol{\Sigma}$*
- *$\boldsymbol{\Sigma} = \biguplus_{\boldsymbol{s} \in \boldsymbol{S}} \boldsymbol{\Sigma}_{\boldsymbol{s}}$ is a symbolic alphabet*
- *$\psi = \{\psi_{\boldsymbol{s}}\}_{\boldsymbol{s} \in \boldsymbol{S}}$ is a family of total surjective functions of the form $\psi_{\boldsymbol{s}} : \Sigma \to \boldsymbol{\Sigma}_{\boldsymbol{s}}$.*

It is required that for every $\boldsymbol{s} \in \boldsymbol{S}$ and $\boldsymbol{a} \in \boldsymbol{\Sigma}_{\boldsymbol{s}}$, $\boldsymbol{s} \cdot \boldsymbol{a} \in \boldsymbol{S} \cup \boldsymbol{R}$ and for any $\boldsymbol{r} \in \boldsymbol{R}$ and $\boldsymbol{a} \in \boldsymbol{\Sigma}$, $\boldsymbol{r} \cdot \boldsymbol{a} \notin \boldsymbol{S} \cup \boldsymbol{R}$. Elements of $\boldsymbol{R}$ are called boundary elements of the tree.

We will use observation tables whose rows are symbolic words and hence an entry in the table will constitute a statement about the inclusion or exclusion of a large *set* of concrete words in the language. We will not ask membership queries concerning all those words, but only for a small representative sample that we call *evidence*.

Definition 5 (Symbolic Observation Table). *A* symbolic observation table *is a tuple $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$ such that*

- *Σ is an alphabet,*
- *$(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi)$ is a finite balanced symbolic Σ-tree (with $\boldsymbol{R}$ being its* boundary*),*
- *E is a subset of Σ^*,*
- *$\boldsymbol{f} : (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E \to \{0, 1\}$ is the symbolic classification function*
- *$\mu : (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E \to 2^{\Sigma^*} - \{\emptyset\}$ is an evidence function satisfying $\mu(\boldsymbol{w}) \subseteq [\boldsymbol{w}]$. The image of the evidence function is prefix-closed: $w \cdot a \in \mu(\boldsymbol{w} \cdot \boldsymbol{a}) \Rightarrow w \in \mu(\boldsymbol{w})$.*

We use, as for the concrete case, $\boldsymbol{f_s} : E \to \{0,1\}$ to denote the partial evaluation of $\boldsymbol{f}$ to some symbolic word $\boldsymbol{s} \in \boldsymbol{S} \cup \boldsymbol{R}$, such that, $\boldsymbol{f_s}(e) = \boldsymbol{f}(\boldsymbol{s} \cdot e)$. Note that the set E consists of *concrete* words but this poses no problem because elements of E are used only to distinguish between states and do not participate in the derivation of the symbolic automaton from the table. The notions of closed, consistent and reduced table are similar to the concrete case.

The set $\boldsymbol{M_T} = (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E$ is called the *symbolic sample* associated with $\boldsymbol{T}$. We require that for each word $\boldsymbol{w} \in \boldsymbol{M_T}$ there is at least one concrete $w \in \mu(\boldsymbol{w})$ whose membership in L, denoted by $f(w)$, is known. The set of such words is called the *concrete sample* and is defined as $M_{\boldsymbol{T}} = \{s \cdot e : s \in \mu(\boldsymbol{s}), \boldsymbol{s} \in \boldsymbol{S} \cup \boldsymbol{R}, e \in E\}$. A table where all evidences of the same symbolic word admit the same classification is called *evidence-compatible*.

Definition 6 (Table Conditions). *A table* $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$ *is*

- *Closed if* $\forall \boldsymbol{r} \in \boldsymbol{R}$, $\exists \boldsymbol{s} = g(\boldsymbol{r}) \in \boldsymbol{S}$, $\boldsymbol{f_r} = \boldsymbol{f_s}$,
- *Reduced if* $\forall \boldsymbol{s}, \boldsymbol{s'} \in \boldsymbol{S}$, $\boldsymbol{f_s} \neq \boldsymbol{f_{s'}}$,
- *Consistent if* $\forall \boldsymbol{s}, \boldsymbol{s'} \in \boldsymbol{S}$, $\boldsymbol{f_s} = \boldsymbol{f_{s'}}$ *implies* $\boldsymbol{f_{s \cdot a}} = \boldsymbol{f_{s' \cdot a}}, \forall \boldsymbol{a} \in \boldsymbol{\Sigma_s}$.
- *Evidence compatible if* $\forall \boldsymbol{w} \in \boldsymbol{M_T}$, $\forall w_1, w_2 \in \mu(\boldsymbol{w}), f(w_1) = f(w_2)$.

When a table $\boldsymbol{T}$ is evidence compatible the symbolic classification function $\boldsymbol{f}$ can be defined for every $\boldsymbol{s} \in (\boldsymbol{S} \cup \boldsymbol{R})$ and $e \in E$ as $\boldsymbol{f}(\boldsymbol{s} \cdot e) = f(s \cdot e)$, $s \in \mu(\boldsymbol{s})$.

Theorem 1 (Automaton from Table). *From a closed, reduced and evidence compatible table* $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$ *one can construct a deterministic symbolic automaton compatible with the concrete sample.*

Proof. Let $\boldsymbol{\mathcal{A}_T} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$ where:

- $Q = \boldsymbol{S}$, $q_0 = \boldsymbol{\epsilon}$
- $F = \{\boldsymbol{s} \in \boldsymbol{S} \mid \boldsymbol{f_s}(\boldsymbol{\epsilon}) = 1\}$
- $\boldsymbol{\delta} : Q \times \boldsymbol{\Sigma} \to Q$ is defined as $\boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{a}) = \begin{cases} \boldsymbol{s} \cdot \boldsymbol{a} & \text{when } \boldsymbol{s} \cdot \boldsymbol{a} \in \boldsymbol{S} \\ g(\boldsymbol{s} \cdot \boldsymbol{a}) & \text{when } \boldsymbol{s} \cdot \boldsymbol{a} \in \boldsymbol{R} \end{cases}$

By construction and like the L^* algorithm, $\boldsymbol{\mathcal{A}_T}$ classifies correctly the symbolic sample. Due to evidence compatibility this holds also for the concrete sample. ⌟

4 The Algorithm

In this section we present a symbolic learning algorithm starting with an intuitive verbal description. From now on we assume that the alphabet is *ordered* and use a_0 to denote its minimal. We assume that the teacher always provides the smallest counter-example with respect to length *and* lexicographic order on Σ^*. Also, when we choose an evidence for a new symbolic word $\boldsymbol{w}$ in a membership query we always take the smallest possible element of $[\boldsymbol{w}]$.

The algorithmic scheme is similar to the concrete L^* algorithm but differs in the treatment of counter-examples and the new concept of evidence compatibility. When the

table is not closed, $\boldsymbol{S} \cup \boldsymbol{R}$ is extended until closure. Then a conjectured automaton $\boldsymbol{\mathcal{A}_T}$ is constructed and an equivalence query is posed. If the answer is positive we are done. Otherwise the teacher provides a counter-example leading possibly to the extension of E and/or $\boldsymbol{S} \cup \boldsymbol{R}$. Whenever such an extension occurs, additional membership queries are posed to fill the table. The table is always kept evidence compatible and reduced except temporarily during the processing of counter-examples.

The learner starts with the symbolic table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$, where $\boldsymbol{\Sigma} = \{\boldsymbol{a_0}\}$, $\boldsymbol{S} = \{\boldsymbol{\epsilon}\}$, $\boldsymbol{R} = \{\boldsymbol{a_0}\}$, $E = \{\epsilon\}$, and $\mu(\boldsymbol{a_0}) = \{a_0\}$. Whenever $\boldsymbol{T}$ is not closed, there is some $\boldsymbol{r} \in \boldsymbol{R}$ such that $f_{\boldsymbol{r}} \neq f_{\boldsymbol{s}}$ for every $\boldsymbol{s} \in \boldsymbol{S}$. To make the table closed we move $\boldsymbol{r}$ from $\boldsymbol{R}$ to $\boldsymbol{S}$ and add to $\boldsymbol{R}$ the word $\boldsymbol{r'} = \boldsymbol{r} \cdot \boldsymbol{a}$, where $\boldsymbol{a}$ is a new symbolic letter with $[\boldsymbol{a}] = \Sigma$, and extend the evidence function by letting $\mu(\boldsymbol{r'}) = \mu(\boldsymbol{r}) \cdot a_0$.

When a counter-example w is presented, it is of course not part of the concrete sample. It admits a factorization $w = u \cdot a \cdot v$, where u is the largest prefix of u such that $u \in \mu(\boldsymbol{u})$ for some $\boldsymbol{u} \in \boldsymbol{S} \cup \boldsymbol{R}$. There are two cases, the second of which is particular to our symbolic algorithm.

1. $\boldsymbol{u} \in \boldsymbol{R}$: Assume that $g(\boldsymbol{u}) = \boldsymbol{s} \in \boldsymbol{S}$ and since the table is reduced, $\boldsymbol{f_u} \neq \boldsymbol{f_{s'}}$ for any other $\boldsymbol{s'} \in \boldsymbol{S}$. Because w is the shortest counter-example, the classification of $\boldsymbol{s} \cdot a \cdot v$ in the automaton is correct (otherwise $s \cdot a \cdot v$, for some $s \in [\boldsymbol{s}]$ would constitute a shorter counter-example) and different from that of $u \cdot a \cdot v$. Thus we conclude that $\boldsymbol{u}$ deserves to be a state and should be added to $\boldsymbol{S}$. To distinguish between $\boldsymbol{u}$ and $\boldsymbol{s}$ we add $a \cdot v$ to E, possibly with some of its suffixes (see [BR04] for a more detailed discussion of counter-example treatment). As $\boldsymbol{u}$ is a new state we need to add its continuations to R. We distinguish two cases depending on a:
   (a) If $a = a_0$ is the smallest element of Σ then a new symbolic letter $\boldsymbol{a}$ is added to $\boldsymbol{\Sigma}$, with $[\boldsymbol{a}] = \Sigma$ and $\mu(\boldsymbol{u} \cdot \boldsymbol{a}) = \mu(\boldsymbol{u}) \cdot a_0$, and the symbolic word $\boldsymbol{u} \cdot \boldsymbol{a}$ is added to $\boldsymbol{R}$.
   (b) If $a \neq a_0$ then *two* new symbolic letters, $\boldsymbol{a}$ and $\boldsymbol{a'}$, are added to $\boldsymbol{\Sigma}$ with $[\boldsymbol{a}] = \{b : b < a\}$, $[\boldsymbol{a'}] = \{b : b \geq a\}$ and $\mu(\boldsymbol{u} \cdot \boldsymbol{a}) = \mu(\boldsymbol{u}) \cdot a_0$, $\mu(\boldsymbol{u} \cdot \boldsymbol{a'}) = \mu(\boldsymbol{u}) \cdot a$. The words $\boldsymbol{u} \cdot \boldsymbol{a}$ and $\boldsymbol{u} \cdot \boldsymbol{a'}$ are added to $\boldsymbol{R}$.
2. $\boldsymbol{u} \in \boldsymbol{S}$: In this case the counter-example indicates that $u \cdot a$ was wrongly assumed to be part of $[\boldsymbol{u} \cdot \boldsymbol{a}]$ for some $\boldsymbol{a} \in \boldsymbol{\Sigma_u}$, and a was wrongly assumed to be part of $[\boldsymbol{a}]$. There are two cases:
   (a) There is some $\boldsymbol{a'} \neq \boldsymbol{a}$ such that the classification of $\boldsymbol{u} \cdot \boldsymbol{a'} \cdot v$ by the symbolic automaton agrees with the classification of $u \cdot a \cdot v$. In this case we just move a and all letters greater than a from $[\boldsymbol{a}]$ to $[\boldsymbol{a'}]$ and no new state is added.
   (b) If there is no such a symbolic letter, we create a new $\boldsymbol{a'}$ with $[\boldsymbol{a'}] = \{b \in [\boldsymbol{a}] : b \geq a\}$ and update $[\boldsymbol{a}]$ to be $[\boldsymbol{a}] - [\boldsymbol{a'}]$. We let $\mu(\boldsymbol{u} \cdot \boldsymbol{a'}) = \mu(\boldsymbol{u}) \cdot a$ and add $\boldsymbol{u} \cdot \boldsymbol{a'}$ to $\boldsymbol{R}$.

A detailed description is given in Algorithm 1 with major procedures in Algorithm 2. A statement of the form $\Sigma = \Sigma \cup \{\boldsymbol{a}\}$ indicates the introduction of a new symbolic letter $\boldsymbol{a} \notin \boldsymbol{\Sigma}$. We use MQ and EQ as shorthands for membership and equivalence queries, respectively. Note also that for every $\boldsymbol{r} \in \boldsymbol{R}$, $\mu(\boldsymbol{r})$ is always a singleton.

We illustrate the behavior of the algorithm on $L = \{a \cdot u : b \leq a < c, u \in \Sigma^*\}$ for two constants $b < c$ in Σ. The table is initialized to $T_0 = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$,

Algorithm 1 The symbolic algorithm

```
1: procedure SYMBOLIC
2:     learned = false
3:     Initialize the table T = (Σ, 𝚺, S, R, ψ, E, f, μ)
4:         𝚺 = {a}; ψ_ε(a) = a, ∀a ∈ Σ
5:         S = {ε}; R = {a}; E = {ε}
6:         μ(a) = {a_0}
7:         Ask MQ on ε and a_0 to fill f
8:     if T is not closed then
9:         CLOSE
10:    end if
11:    repeat
12:        if EQ(A_T) then                          ▷ A_T is correct
13:            learned = true
14:        else                                     ▷ A counter-example w is provided
15:            M = M ∪ {w}
16:            COUNTER-EX(w)                        ▷ Process counter-example
17:        end if
18:    until learned
19: end procedure
```

where $\boldsymbol{\Sigma} = \{\boldsymbol{a}_0\}$, $\mu(\boldsymbol{a}_0) = \{a_0\}$, $\boldsymbol{S} = \{\epsilon\}$, $E = \{\epsilon\}$, $\boldsymbol{R} = \{\boldsymbol{a}_0\}$ and $\psi = \{\psi_\epsilon\}$ with $\psi_\epsilon(a) = \boldsymbol{a}_0, \forall a \in \Sigma$. We ask membership queries to learn $f(\epsilon)$ and $f(a_0)$. Table T_0, shown in Fig. 3, is closed, reduced and evidence compatible and its related hypothesis automaton $\boldsymbol{\mathcal{A}}_0$ consists of only one rejecting state, as shown in Fig. 4. The teacher responds to this conjecture by the counter-example $+b$. Since $b \notin \mu(\boldsymbol{a}_0)$ and $\epsilon \in \boldsymbol{S}$, we are in Case 2-(b) of the counter-example treatment, where there is no symbolic word that classifies b correctly. We create a new symbolic letter $\boldsymbol{a}_1$ with $\mu(\boldsymbol{a}_1) = \{b\}$ and modify ψ_ϵ to $\psi_\epsilon(a) = \boldsymbol{a}_0$ when $a < b$ and $\psi_\epsilon(a) = \boldsymbol{a}_1$ otherwise. The derived table T_1 is not closed since for $\boldsymbol{a}_1 \in \boldsymbol{R}$ there is no element $\boldsymbol{s} \in \boldsymbol{S}$ such that $\boldsymbol{f}_{\boldsymbol{a}_1} = \boldsymbol{f}_{\boldsymbol{s}}$. To close the table we move $\boldsymbol{a}_1$ from $\boldsymbol{R}$ to $\boldsymbol{S}$ and introduce a new symbolic letter $\boldsymbol{a}_2$ to represent the continuations of $\boldsymbol{a}_1$. We define $\psi_{\boldsymbol{a}_1}$ with $\psi_{\boldsymbol{a}_1}(a) = \boldsymbol{a}_2$ for all $a \in \Sigma$, $\mu(\boldsymbol{a}_1 \cdot \boldsymbol{a}_2) = \{b \cdot a_0\}$ and add the symbolic word $\boldsymbol{a}_1 \cdot \boldsymbol{a}_2$ to $\boldsymbol{R}$. We ask membership queries for the missing words and construct a new observation table T_2.

This table is closed and reduced, resulting in a new hypothesis automaton $\boldsymbol{\mathcal{A}}_2$. The counter-example provided by the teacher is $-c$. This is case 2-(a) of the counter-example treatment as there exists a symbolic letter $\boldsymbol{a}_0$ that agrees with the classification of c. We move c and all elements greater than it from $[\boldsymbol{a}_1]$ to $[\boldsymbol{a}_0]$, that is, $\psi_\epsilon(a) = \boldsymbol{a}_0$ when $a < b$, $\psi_\epsilon(a) = \boldsymbol{a}_1$ when $b \leq a < c$ and $\psi_\epsilon(a) = \boldsymbol{a}_3$ otherwise. Table T_3 is closed, reduced and evidence compatible leading to the hypothesis automaton $\boldsymbol{\mathcal{A}}_3$ for which $-ab$ is a counter-example where $a \in \mu(\boldsymbol{a}_0)$ and $\boldsymbol{a}_0 \in \boldsymbol{R}$. Thus we are now in case 1 and since the counter-example is considered to be the shortest, $\boldsymbol{a}_0$ is a new state of the automaton, different from ϵ. We move $\boldsymbol{a}_0$ to $\boldsymbol{S}$ and add a new symbolic letter $\boldsymbol{a}_4$ to $\boldsymbol{\Sigma}$, which represents the transition from $\boldsymbol{a}_0$, with $\mu(\boldsymbol{a}_4) = \{a_0\}$. Now $\psi_{\boldsymbol{a}_0}(a) = \boldsymbol{a}_4$ and $\psi_{\boldsymbol{a}_1}(a) = \boldsymbol{a}_2$ for all $a \in \Sigma$. However the obtained table T_4 is not reduced since

Procedures 2 Closing the table and processinging counter-examples

procedure CLOSE ▷ Make the table closed
  while $\exists \boldsymbol{r} \in \boldsymbol{R}$ such that $\forall \boldsymbol{s} \in \boldsymbol{S}$, $\boldsymbol{f_r} \neq \boldsymbol{f_s}$ **do**
    $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}\}$; $\psi' = \psi \cup \{\psi_{\boldsymbol{r}}\}$ with $\psi_{\boldsymbol{r}}(a) = \boldsymbol{a}, \forall a \in \Sigma$
    $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{r}\}$; $\boldsymbol{R'} = (\boldsymbol{R} - \{\boldsymbol{r}\}) \cup \{\boldsymbol{r} \cdot \boldsymbol{a}\}$
    $\mu(\boldsymbol{r} \cdot \boldsymbol{a}) = \mu(\boldsymbol{r}) \cdot a_0$
    Ask MQ for all words in $\{\mu(\boldsymbol{r} \cdot \boldsymbol{a}) \cdot e : e \in E\}$
    $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma'}, \boldsymbol{S'}, \boldsymbol{R'}, \psi', E, \boldsymbol{f'}, \mu')$
  end while
end procedure

procedure COUNTER-EX(w) ▷ Process counter-example
  Find a factorization $w = u \cdot a \cdot v$, $a \in \Sigma$, $u, v \in \Sigma^*$ such that
    $\exists \boldsymbol{u} \in \boldsymbol{M_T}$, $u \in \mu(\boldsymbol{u})$ and $\forall \boldsymbol{u'} \in \boldsymbol{M}$, $u \cdot a \notin \mu(\boldsymbol{u'})$
  if $\boldsymbol{u} \in \boldsymbol{R}$ **then**
    if $a = a_0$ **then** ▷ Case 1(a)
      $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}\}$; $\psi' = \psi \cup \{\psi_{\boldsymbol{u}}\}$, with $\psi_{\boldsymbol{u}}(\sigma) = \boldsymbol{a}, \forall \sigma \in \Sigma$
      $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{u}\}$; $\boldsymbol{R'} = (\boldsymbol{R} - \{\boldsymbol{u}\}) \cup \{\boldsymbol{u} \cdot \boldsymbol{a}\}$; $E' = E \cup \{\text{suffixes of } a \cdot v\}$
      $\mu(\boldsymbol{u} \cdot \boldsymbol{a}) = \mu(\boldsymbol{u}) \cdot a_0$
      Ask MQ for all words in $\{\mu(\boldsymbol{u} \cdot \boldsymbol{a}) \cdot e : e \in E'\}$
    else ▷ Case 1(b)
      $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}, \boldsymbol{a'}\}$
      $\psi' = \psi \cup \{\psi_{\boldsymbol{u}}\}$, with $\psi_{\boldsymbol{u}}(\sigma) = \begin{cases} \boldsymbol{a} & \text{if } \sigma < a \\ \boldsymbol{a'} & \text{otherwise} \end{cases}$
      $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{u}\}$; $\boldsymbol{R'} = (\boldsymbol{R} - \{\boldsymbol{u}\}) \cup \{\boldsymbol{u} \cdot \boldsymbol{a}, \boldsymbol{u} \cdot \boldsymbol{a'}\}$; $E' = E \cup \{\text{suffixes of } a \cdot v\}$
      $\mu(\boldsymbol{u} \cdot \boldsymbol{a}) = \mu(\boldsymbol{u}) \cdot a_0$; $\mu(\boldsymbol{u} \cdot \boldsymbol{a'}) = \mu(\boldsymbol{u}) \cdot a$
      Ask MQ for all words in $\{(\mu(\boldsymbol{u} \cdot \boldsymbol{a}) \cup \mu(\boldsymbol{u} \cdot \boldsymbol{a'})) \cdot e : e \in E'\}$
    end if
    $\boldsymbol{T'} = (\Sigma, \boldsymbol{\Sigma'}, \boldsymbol{S'}, \boldsymbol{R'}, \psi', E', \boldsymbol{f'}, \mu)$
  else ▷ Case 2(a),(b)
    Find $\boldsymbol{a} \in \boldsymbol{\Sigma_u}$ such that $a \in [\boldsymbol{a}]$
    if there is no $\boldsymbol{a'} \in \boldsymbol{\Sigma} : f_{u \cdot a} = f_{\mu(\boldsymbol{u} \cdot \boldsymbol{a'})}$ on E **then**
      $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a'}\}$; $\boldsymbol{R'} = \boldsymbol{R} \cup \{\boldsymbol{u} \cdot \boldsymbol{a'}\}$
      $\mu(\boldsymbol{u} \cdot \boldsymbol{a'}) = \mu(\boldsymbol{u}) \cdot a$
      Ask MQ for all words in $\{\mu(\boldsymbol{u} \cdot \boldsymbol{a'}) \cdot e : e \in E\}$
    end if
    $\psi_{\boldsymbol{u}}(\sigma) = \begin{cases} \psi_{\boldsymbol{u}}(\sigma) & \text{if } \sigma \notin [\boldsymbol{a}] \\ \boldsymbol{a} & \text{if } \sigma \in [\boldsymbol{a}] \text{ and } \sigma < a \\ \boldsymbol{a'} & \text{otherwise} \end{cases}$
    $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma'}, \boldsymbol{S}, \boldsymbol{R'}, \psi, E, \boldsymbol{f'}, \mu)$
  end if
  if $\boldsymbol{T}$ is not closed **then**
    CLOSE
  end if
end procedure

$\boldsymbol{f}_\epsilon(e) = \boldsymbol{f}_{\boldsymbol{a}_0}(e)$ for all $e \in E$. We add experiment b to E and fill the gaps using membership queries, resulting in table T_5 which is closed, reduced and evidence compatible. The derived automaton $\boldsymbol{\mathcal{A}}_5$ is the right one and the algorithm terminates.

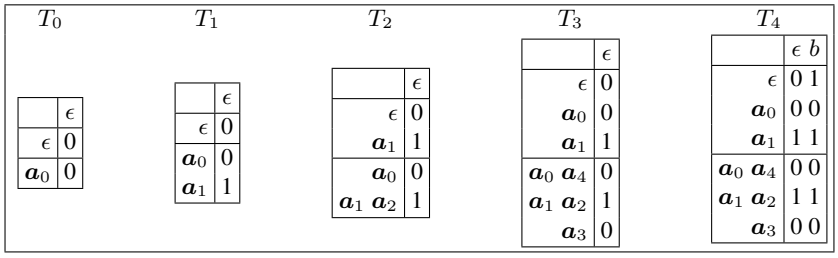

Fig. 3. Symbolic observation tables for $(b \leq a < c) \cdot \Sigma^*$

It is easy to see that for large alphabets our algorithm is much more efficient than L^*. For example, when $\Sigma = \{1..100\}$, $b = 20$ and $c = 50$, the L^* algorithm will need around 400 queries while ours will ask less than 10. The symbolic algorithm is influenced not by the size of the alphabet but by the resolution (partition size) with which we observe it. Fig. 5 shows a larger automaton over the same alphabet learned by our procedure.

5 Discussion

We have defined a generic algorithmic scheme for automaton learning, targeting languages over large alphabets that can be recognized by finite symbolic automata having a modest number of states and transitions. Some ideas similar to ours have been proposed for the particular case of parametric languages [BJR06] and recently in a more general setting [HSM11,IHS13] including partial evidential support and alphabet refinement during the learning process.[2]

The genericity of the algorithm comes from the semantic approach (alphabet partitions) but of course, each and every domain will have its own semantic and syntactic specialization in terms of the size and shape of the alphabet partitions. In this work we have implemented an instantiation of this scheme for the alphabet $\Sigma = (\mathbb{N}, \leq)$ and the adaptation to real numbers is immediate. When dealing with numbers, the partition into a finite number of intervals (and convex sets in higher dimensions) is very natural and used in many application domains ranging from quantization of sensor readings to income tax regulations. It will be interesting to compare the expressive power and succinctness of symbolic automata with other approaches for representing numerical time series and to compare our algorithm with other inductive inference techniques for sequences of numbers.

[2] Let us remark that the modification of partition boundaries is not always a refinement in the precise mathematical sense of the term.

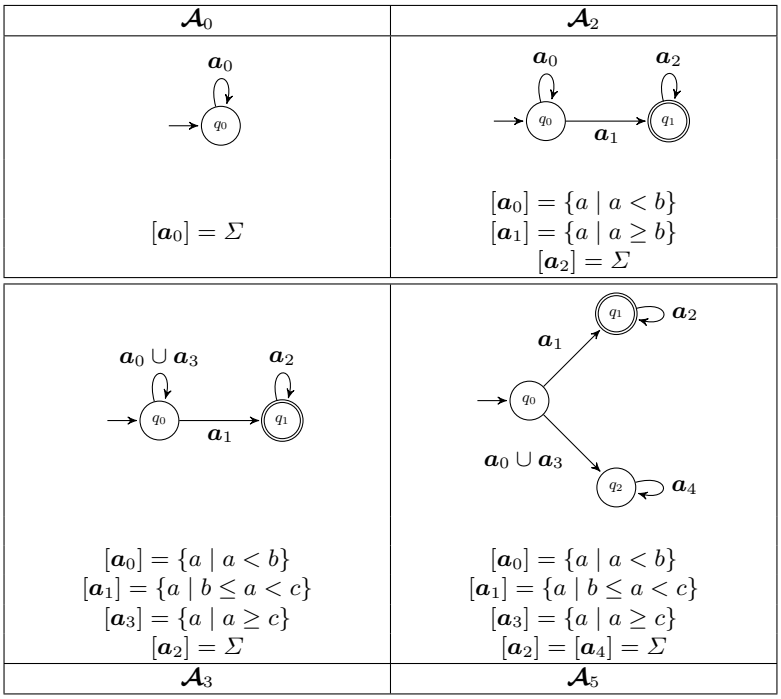

Fig. 4. Hypothesis automata for $(b \leq a < c) \cdot \Sigma^*$

As a first excursion into the domain, we have made quite strong assumptions on the nature of the equivalence oracle, which, already for small alphabets, is a bit too strong and pedagogical to be realistic. We assumed that it provides the shortest counter-example and also that it chooses always the minimal available concrete symbol. We can relax the latter (or both) and replace the oracle by random sampling, as already proposed in [Ang87] for concrete learning. Over large alphabets, it might be even more appropriate to employ probabilistic convergence criteria a-la *PAC learning* [Val84] and be content with a correct classification of a large fraction of the words, thus tolerating imprecise tracing of boundaries in the alphabet partitions. This topic, as well as the challenging adaptation of our framework to languages over Boolean vectors are left for future work.

Acknowledgement: This work was supported by the French project EQINOCS (ANR-11-BS02-004). We thank Peter Habermehl, Eugene Asarin and anonymous referees for useful comments and pointers to the literature.

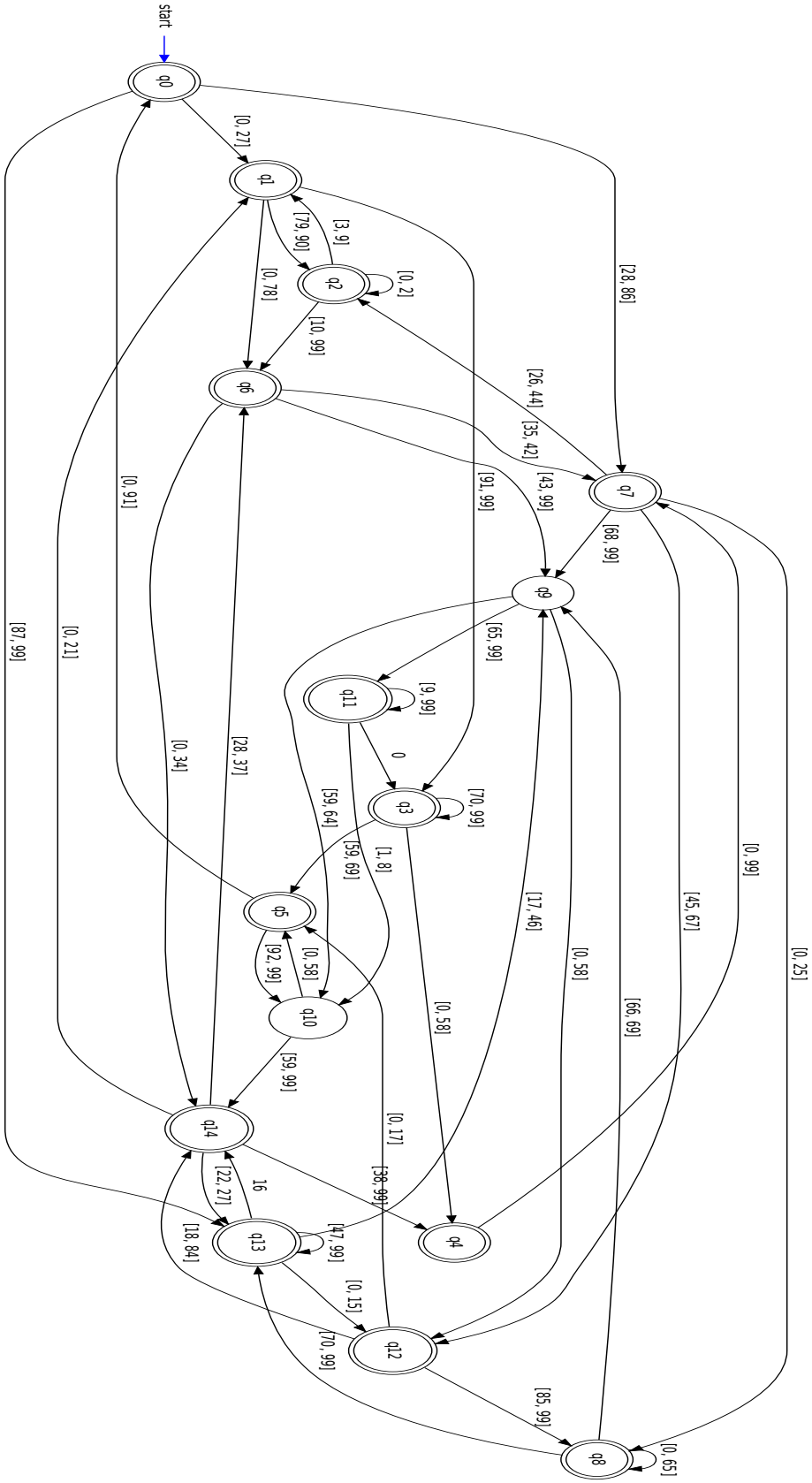

Fig. 5. An automaton learned by our procedure using 418 membership queries, 27 equivalence queries with a table of size 46×11.

References

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[BJR06] Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines with parameters. In *FASE*, pages 107–121. Springer, 2006.

[BLP10] Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In *AMW*, 2010.

[BR04] Therese Berg and Harald Raffelt. Model checking. In *Model-Based Testing of Reactive Systems*, pages 557–603, 2004.

[DlH10] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[DR95] Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995.

[Gol72] E. Mark Gold. System identification via state characterization. *Automatica*, 8(5):621 – 636, 1972.

[HSJC12] Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In *VMCAI*, pages 251–266, 2012.

[HSM11] Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In *VMCAI*, pages 263–277, 2011.

[IHS13] Malte Isberner, Falk Howar, and Bernhard Steffen. Inferring automata with state-local alphabet abstractions. In *NASA Formal Methods*, pages 124–138, 2013.

[KF94] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[Moo56] Edward F Moore. Gedanken-experiments on sequential machines. In *Automata studies*, volume 34 of *Annals of Mathematical Studies*, pages 129–153. Princeton, 1956.

[MP95] Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[VHL+12] Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Björner. Symbolic finite state transducers: algorithms and applications. In *POPL*, pages 137–150, 2012.